

**Lab report**

|  |  |
| --- | --- |
| **Course**: | Class Libraries and Data Structures |
| **Semester**: | 1st semester of the academic year **2020-2021** |
| **Major**: | Software Engineering |
| **Class**: | 2019 |
| **Student Name**: | 冯春霖 |
| **Student ID:** | 222019321062074 |
| **Teacher:** | ZHAO, Hengjun (赵恒军) |

**School of Computer and Information Science**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | | Recursion and Backtrack | | | |
| Date | | Dec 10，2020 | Type | | □Confirmatory  √ Design  □Comprehensive |
| 1. **Objective & Requirements**    1. Familiarize you with the basic idea of recursion    2. Understand the design principles of recursive functions and can design your own recursive functions    3. Understand the principle of backtracking and can design a backtracking algorithm to solve a practical problem | | | | | |
| 1. **Experimental environment (**platform and software**)**   Windows 7 (or higher versions) + Visual Studio 2010 (or higher versions) | | | | | |
| 1. **Experimental content and design** (Main Content, Procedure, Codes and Results)   Task 1:  Use recursion to output the n-th Fibonacci number.  Task 2:  A chessboard has eight rows and eight columns. In the game of chess, the queen is the most powerful piece: she can attack any piece in her row, any piece in her column, and any piece in either of her diagonals. Develop and validate a program to place eight queens on a chessboard in such a way that no queen is under attack from any other queen. (Hint: use recursion and backtrack)  Task 1:  Code:  #include <iostream>  int Fibo(int n)  {  if (n == 1 || n == 2)  return 1;  else  return (Fibo(n - 1) + Fibo(n - 2));  }  int main()  {  int n;  std::cout << "Please input number n: ";  std::cin >> n;  std::cout << "Fibo(" << n << ") = " << Fibo(n) << std::endl;  return 0;  }  Screenshot of the program results:    Task 2:  Code:  #include <iostream>  using namespace std;  int board[8][8]{ 0 };  void printBoard()  {  for (int i = 0; i < 8; i++)  {  for (int j = 0; j < 8; j++)  cout << board[i][j] << ' ';  cout << endl;  }  }  bool isValid(int row, int col)  {  for (int i = 0; i < row; i++)  {  int invalidCol;  for (int j = 0; j < 8; j++)  if (board[i][j])  invalidCol = j;  if (col == invalidCol)  return false;  if ((i + invalidCol) == (row + col))  return false;  if ((i - invalidCol) == (row - col))  return false;  }  return true;  }  void goBack(int row)  {  for (int i = 0; i < 8; i++)  board[row][i] = 0;  return;  }  bool backTrack(int row)  {  for (int col = 0; col < 8; col++)  if (isValid(row, col))  {  board[row][col] = 1;  if (row == 7)  {  printBoard();  cout << endl;  exit(0);  }  backTrack(row + 1);  goBack(row);  }  }  int main()  {  backTrack(0);  return 0;  }  Screenshot of the program results:    Improvements to backTrack():  Execute goBack() after each execution, can get different placement results, add the global variable *counts* for counting, can get a total of 92 possible placement methods.  Screenshot of the improved program results: | | | | | |
| 1. **Result analysis and discussion**（Analysis of experimental results and summing up the harvest and the existing problems）   Through this experiment I have gained a better understanding of recursion and backtracking algorithms. This experiment was not very difficult for me because I had a foundation in algorithms before and I was able to improve the program I wrote to achieve a higher goal. | | | | | |
| Comments & Evaluation | Content & Design (A-E) | | |  | |
| Procedure & Codes (A-E) | | |  | |
| Results (A-E) | | |  | |
| Analysis & Discussion (A-E) | | |  | |
| Score (A-E):  Feedback comments: | | | | |